N95- 23744

DTS: Building Custom, Intelligent Schedulers

Othar Hansson and Andrew Mayer
Heuristicrats Research, Inc.
1678 Shattuck Avenue, Suite 310
Berkeley, CA 94709-1631
(510) 845-5810, Fax: (510) 845-4405
{othar, mayer}@heuristicrat.com

KEY WORDS AND PHRASES

Decision theory, heuristic search, optimization, scheduling, user interface.

ABSTRACT

DTS is a *decision-theoretic scheduler*, built on top of a flexible toolkit—this paper focuses on how the toolkit might be reused in future NASA mission schedulers. The toolkit includes a user-customizable scheduling interface, and a "Just-For-You" optimization engine.

The customizable interface is built on two metaphors: objects and dynamic graphs. Objects help to structure problem specifications and related data, while dynamic graphs simplify the specification of graphical schedule editors (such as Gantt charts). The interface can be used with any "back-end" scheduler, through dynamically-loaded code, interprocess communication, or a shared database.

The "Just-For-You" optimization engine includes user-specific utility functions, automatically compiled heuristic evaluations, and a postprocessing facility for enforcing scheduling policies. The optimization engine is based on BPS, the Bayesian Problem-Solver [1,2], which introduced a similar approach to solving single-agent and adversarial graph search problems.

DTS SYSTEM OVERVIEW

The Decision-Theoretic Scheduler, DTS, is designed to support scheduling of over-subscribed, long-running projects. DTS is literally implemented as a program in a specialized language for the design of scheduling and optimization systems. This DTS Customization Language (DCL) is implemented on top of the public-domain TCL/Tk system [3].

DTS has been designed for science-planning on NASA missions. We are preparing to deploy the system as one component of a cost-reduction program within the Extreme Ultraviolet Explorer mission of the Center for Extreme Ultraviolet Astrophysics at the University of California, Berkeley [4].

We have explicitly designed DTS to be customizable by users, and thus transferrable to other missions. An easily customized scheduling system can reduce costs by eliminating the mission-specific paperwork and "workarounds" that result when a system does not address a scheduling scenario completely.

To reduce mission costs further, we have designed DTS so that such extensions can be made quickly and without corrupting existing code or functionality. For example, the current DTS interface provides much of the functionality of commercial project scheduling tools, but is implemented in under 7000 lines of DCL code. User modifications—such as an import "filter" for a pre-existing file format, or a specialized report writer—typically require only a few dozen lines of DCL code. Because DCL code is interpreted, programming errors are safely trapped.

Behind the scenes, the DTS "back-end" contains a sophisticated constraint-satisfaction search engine for use in automated scheduling. The use of decision theory permits user preferences and requirements to be modeled in a

mathematically coherent way. The result is that DTS can typically find near-optimal solutions to the user's actual problem, with optimality measured in the user's terms. Many existing scheduling techniques restrict both the definition of optimality and the representation of the problem: the user is forced to use a system that provides a quasi-optimal solution to an approximation of the problem.

Our research goal in the DTS back-end has been to provide a rich representation for problems and preferences, and still find near-optimal solutions through the use of compilation, learning and decision-theoretic search.

In this paper, we describe customization in both the front-end and back-end, and then conclude with a description of future plans for applying DTS to NASA missions.

USER INTERFACE CUSTOMIZATION

The DTS interface uses objects and dynamic graphs to support customization.

All data in the system is represented within an object hierarchy. The hierarchy includes Task objects, Constraint objects, etc., as you would expect. These basic objects can be subclassed, or specialized, for the needs of an individual application: in the NASA version of DTS, an Observation object represents each Task that is an astronomical observation.

The system also includes "management" information objects such as (astronomical) Targets, (scientific) Proposals, and Principal Investigators. This information is linked to "problem" information such as tasks by the use of cross-reference attributes. For example, each Observation has an attribute named Target that is a cross-reference.

The DTS interface is centered on an object browser (Figure 2). Customization begins by defining a new object class, or redefining an existing object class. Each object class has an associated form, used to display and edit object instances in the browser. A simple default form is inferred from the "type" of each attribute (String, Date, etc.).

More complex forms require the use of DCL code. Figure 2 shows the form for a TemporalConstraint instance. This is the most complicated form in the system, but it requires only 40 lines to produce a specialized display

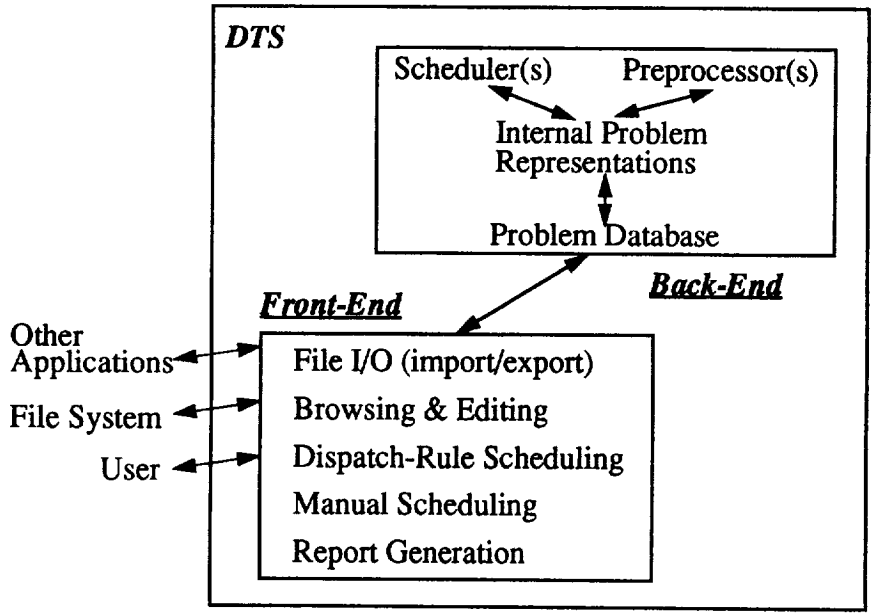

Figure 1. Overview of DTS System Architecture.

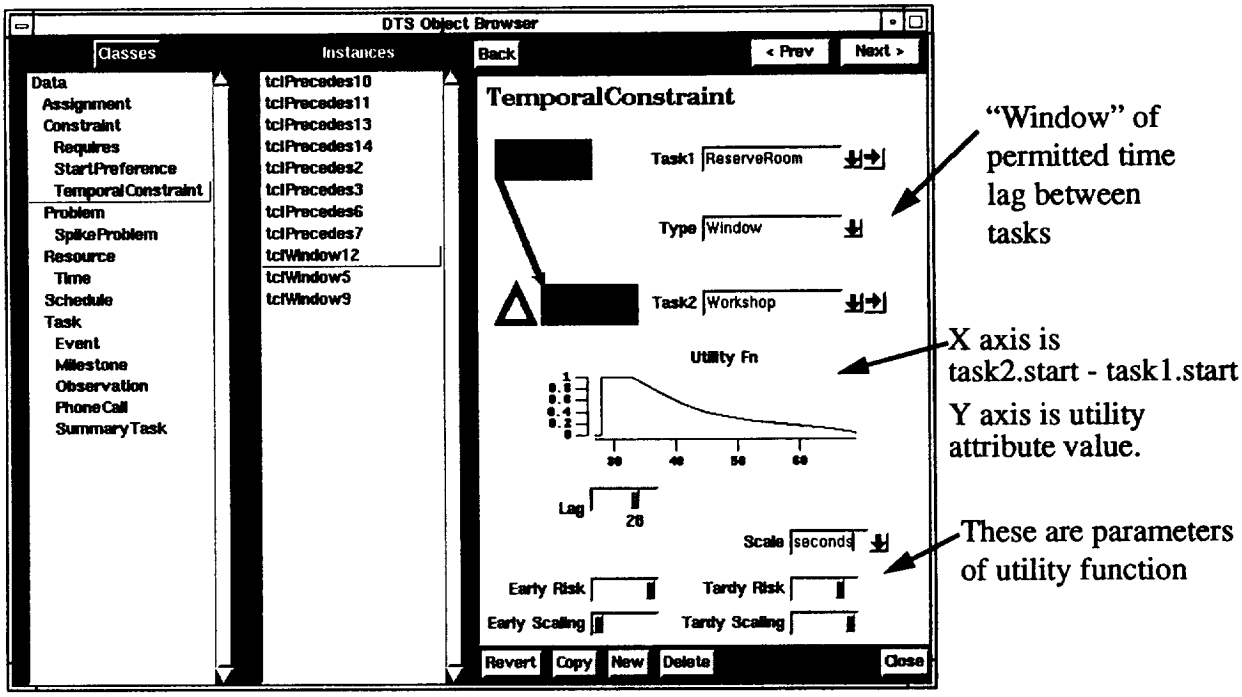

Figure 2. Example Customized Form in the Object Browser.

for a number of interrelated attributes. Like most binary constraints, the temporal constraint has two task parameters. In addition, for constraints of type "window," a utility function is defined by the parameters at the bottom of the form. These parameters are "animated" in a utility graph. Finally, each type of constraint has an associated graphical mnemonic (the upper left of the form), which reminds the user of the nature of the constraining relationship.

The second major mechanism in the DTS user interface is the dynamic graph. Dynamic graphs are editable "views" of a number of objects, built using an X-Y graphing metaphor. For example, a typical Gantt chart is an X-Y plot of tasks (Y), using their start time and duration (X). The DTS dynamic graph permits views such as Gantt charts, PERT charts, constraint matrices and resource histograms to be specified easily. These graphs are dynamic in that callbacks can be associated with user actions (e.g., mouse events), and defined to modify the underlying data appropriately.

Each of the basic views implemented thus far has required approximately 250 lines of DCL code for layout and callbacks. Application-specific views (such as augmented Gantt charts, statistical summaries, etc.) should be implementable with similar effort.

OPTIMIZER CUSTOMIZATION

The DTS back-end includes C++ routines, callable through DCL, that perform basic preprocessing and scheduling tasks. This optimization engine uses decision-theoretic search mechanisms developed by the authors in previous and ongoing work with the Bayesian Problem-Solver [1,2,5].

The use of decision theory [6,7,8] enables the engine to guide its search by user-specific utility functions, in addition to heuristic evaluation functions. Many existing schedulers use heuristic functions alone, but heuristic functions can confuse the role of schedule evaluation (utility) and search control (heuristics).

DTS collects statistics that relate heuristic evaluations to attributes of the utility function. Because these statistics relate to inputs rather than outputs of the utility function, the func-

tion itself can be modified without invalidating the statistics that have been gathered. The use of statistical estimation and probabilistic inference in DTS also permits multiple heuristic evaluations to be combined to focus the search more effectively. For example, a general-purpose constraint-satisfaction heuristic might be coupled with a domain-specific heuristic [5].

In an early phase of development, we found that the costs of state generation and heuristic evaluation were a significant bottleneck to the development of sophisticated scheduling search control. DTS thus also employs an experimental compilation mechanism that derives a specialized data structure for search tree "states" from a formal specification of the heuristic function. Hand-coding of such data structures reduces the overall cost of search significantly, and we anticipate that the automation of these data structures will permit these benefits to be achievable for users relying on domain-specific heuristics. Hansson [9] describes the compilation mechanism in more detail.

Finally, the use of DCL permits a user to code a secure "audit" or "checker" routine to validate a finished schedule before execution, or to enforce certain scheduling policies that are hard to represent within the system.

Along with other DTS features, these three mechanisms—decision-theoretic search with user-specific utility functions, data structure compilation for fast heuristic evaluation, and postprocessing for schedule validity—have been designed to ensure that DTS finds solutions to the user's *real* problem with a minimum of search cost.

CONCLUSION

We are presently customizing DTS for possible use within current and future NASA missions (including EUVE and CASSINI), and collaborating with NASA researchers to reuse the DTS interface on top of their schedulers.

We feel that the customizability of DTS can permit future NASA missions to exploit "economies of reuse" and "economies of fidelity." Economies of reuse are well-known: they result when development costs are cut by reusing flexible software.

Economies of fidelity result when a system can be made to solve a large portion of an application task, without a great degree of simplification. Many search and optimization frameworks require the user to simplify or abstract their problem into a restricted modelling language. This increases the cost of using such systems, and reduces the benefits: the solutions found are not always executable, let alone near-optimal, solutions to the real problem. On the other hand, systems like DTS, and Muscettola's HSTS [10], attempt to provide a richer framework for modeling the problem. DTS focuses on preference modeling, while HSTS focuses on constraint and state-variable modeling. We anticipate that compilation and learning techniques will permit these rich representations to be searched efficiently.

BIBLIOGRAPHY

[1] Hansson, O. and A. Mayer. "Heuristic Search as Evidential Reasoning." In *Proc. of the Fifth Workshop on Uncertainty in Artificial Intelligence*, Windsor, Ontario, August 1989.

[2] Mayer, A. *Rational Search.* Ph.D. Dissertation, Univ. of California, 1994.

[3] Ousterhout, J. *TCL and the Tk Toolkit.* Addison-Wesley, Reading, MA, 1994.

[4] Center for EUV Astrophysics. *EUVE Guest Observer Program Handbook.* Appendix G of NASA NRA 92-OSSA-5. Berkeley, Jan. 1992.

[5] Hansson, O. and A. Mayer. "Decision-Theoretic Control of Artificial Intelligence Scheduling Systems." HRI Technical Report No. 90-1/06.04/5810, Berkeley, CA, September 1991.

[6] Pearl, J. *Probabilistic Reasoning in Intelligent Systems.* Morgan Kaufmann, San Mateo, CA, 1988.

[7] Savage, L. J. *The Foundations of Statistics.* Dover, New York, 1972.

[8] von Neumann, J. and O. Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.

[9] Hansson, O. *Bayesian Problem-Solving applied to Scheduling.* Ph.D. Dissertation, Univ. of California, 1994.

[10] Muscettola, N. *HSTS-DDL Manual,* NASA Ames Research Center, Code FIA, March 1994.